Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dejan Obrez

Ogrodje mobilne aplikacije mFRI

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvijte ogrodje mobilne aplikacije mFRI za Windows platformo. Ogrodje naj bo zasnovano tako, da bo dodajanje novih funkcionalnosti enostavno. V okviru ogrodja razvijte module, ki bodo študentom omogočali dostop do storitev Trola, Bicikelj in Urnik. V okviru te modulov v največji možni meri uporabite že obstoječe storitve in vire, do katerih je dostop možen. Uporabite tehnologije UWP in WPF.

Rad bi se zahvalil predvsem družini, ki mi je tekom celotnega študija stala ob strani. Zahvaljujem se tudi mentorju doc. dr. Roku Rupniku.

Kazalo

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
XAML	extensible application markup language	razširljiv označevalni jezik
PHP	Personal Home Page Tools	orodja za osebno spletno stran
JSON	JavaScript Object Notation	zapis objekta JavaScript
MVVM	Model–view–viewmodel	model-pogled-modelPogled
WPF	Windows Presentation Foundation	sistem za izdelavo grafičnih vmesnikov
PLC	Portable Class Library	prenosna razredna knjižnica
ARM	Advanced RISC Machine	napreden RISC-stroj
RISC	reduced instruction set computing	računanje z zmanjšanjim številom ukazov
UWP	Universal Windows Platform	univerzalna platforma Windows
SoC	Separation of concerns	ločevanje delov kode
API	Application programming interface	programski vmesnik
SDK	Software development kit	skupek orodij za razvijalca
IoT	Internet of things	povezava vsakodnevnih naprav z internetom
LINQ	Language Integrated Query	jezikovno integrirana poizvedba

Povzetek

Naslov: Ogrodje mobilne aplikacije mFRI

Namen te diplomske naloge je razviti funkcionalno ogrodje aplikacije, ki ga bodo lahko študentje te fakultete uporabljali na vsakodnevni ravni kot pomoč pri študiju ali nasploh življenju v Ljubljani.

V diplomskem delu sem temeljito predstavil vse tehnologije, s katerimi sem se pri razvoju srečal, arhitekturo, modularnost in nasploh idejo, kako je aplikacija sestavljena, ter tudi, kako deluje.

Ogrodje je prav tako namenjeno študentom, ki bodo želeli nadaljevati razvoj na tej aplikaciji, zato je pomembno, da je čim bolj robustno, preprosto za dodajanje tako novih funkcionalnosti na poslovnem delu aplikacije kot na uporabniškem vmesniku.

Ključne besede: Mobilna aplikacija, Windows Phone, Ogrodje, Windows 8.1 Universal, UWP, MVVM.

Abstract

Title: mFRI Mobile Application Framework

The purpose of this graduation thesis is to develop a functional framework of application, which will be used by the students of this faculty in everyday life as a help in studies or generally in life in Ljubljana.

In the graduation thesis, I have thoroughly presented all technologies, which I encountered during

development: architecture, modularity, and the idea itself of how the application is designed and also how it works.

The framework is also intended for students, who will want to continue the development in this application. Therefore, it is important that it is as robust as possible, as well as simple for adding new functionalities in business part of application as well in the user interface part.

Keywords: Mobile application, Windows Phone, Framework, Windows 8.1 Universal, UWP, MVVM.

Poglavje 1

Uvod

Živimo v dobi, v kateri je računalnik v obliki pametnega telefona postal del našega vsakdana, zato ni čudno, da se je povečala potreba po razvoju mobilnih aplikacij, s katerimi lahko dostopamo do informacij kjerkoli in kadarkoli. V tem duhu so tudi postavljene trenutne smernice v informacijskem svetu, kjer nam že skoraj vsaka spletna stran ponuja svojo mobilno aplikacijo za lažji dostop do njihovih vsebin. Temu trendu želimo slediti tudi na Fakulteti za računalništvo in informatiko, zato sem se za diplomsko delo odločil pripraviti funkcionalno ogrodje mobilne aplikacije za Windows Phone, s katerim bi študentom olajšali obiskovanje fakultete in posredovali informacije za njihove študijske ter izvenštudijske dejavnosti.

Cilj te diplomske naloge je pripraviti čim boljše ogrodje mobilne aplikacije. Vsebuje najnovejše tehnologije in prakse, ki se uporabljajo tudi v profesionalnem razvoju aplikacij. Ogrodje bo vsebovalo že precej funkcionalnosti, kar pomeni, da bodo študentje lahko aplikacijo že uporabljali. Eden izmed ciljev je tudi razvoj takšnega ogrodja, ki bo študentom, če bodo to hoteli, omogočal enostaven nadaljnji razvoj in implementacijo novih funkcionalnosti.

V nadaljevanju tega dokumenta bom opisal delovni tok, tehnologije, arhitekturo in implementirane funkcionalnosti, s katerimi sem dosegel integriteto posameznih komponent, enostavno brisanje, dodajanje ter spreminjanje

komponent in videza aplikacije.

V nadaljevanju tega dokumenta bom podrobno opisal tako uporabljene tehnologije kot tudi sorodne in naredil primerjavo med njimi. Omenil bom tudi arhitekturni vzorec MVVM, ki je glavno gonilo naše aplikacije, saj je njegova izbira bistveno vplivala na celoten razvoj ogrodja. Beseda pa bo nanesla tudi na sam delovni tok aplikacije, implementacijo modulov in z njimi funkcionalnosti ter integriteto, ki jo prinašajo. Dotaknili pa se bomo tudi uporabniškega vmesnika, večjezičnosti ...

Poglavje 2

Uporabljene in sorodne tehnologije

Pri izdelavi diplomskega dela sem uporabil naslednje tehnologije:

- **C Sharp** je objektno orientiran programski jezik, ki ga je razvil Microsoft za delovanje znotraj okolja .NET. Microsoftov cilj je bil poenostavitev izmenjave informacij med spletnimi storitvami in omogočiti razvijalcem razvoj visoko prenosnih mobilnih naprav.
- **XAML** je Microsoftova verzija jezika XML za oblikovanje uporabniškega vmesnika. Omogoča nam definiranje in oblikovanje posameznih elementov GUIja, vezanje podatkov iz objektov, proženje akcij ...
- **JSON** je enostaven format za izmenjavo podatkov. Jezik je človeku berljiv, prav tako je enostaven za strojno obdelavo. Uporablja se predvsem pri prenosu podatkov med serverjem in aplikacijo kot alternativa XML-ju.
- **PHP**je preprost odprtokodni skriptni jezik, namenjen predvsem razvoju spletnih aplikacij, in ga z lahkoto vključimo v HTML. Izvaja se na strani strežnika in je bil v diplomskem delu uporabljen predvsem za prepisovanje podatkov, ki jih dobimo s spleta, ter njihovo posredovanje naši mobilni aplikaciji.

- **SQLite** je relacijska baza in hkrati sistem za upravljanje s podatkovnimi bazami. SQlLite je bolj ali manj samozadosten, kar pomeni, da ne potrebuje velike podpore iz drugih knjižnic za delovanje, poleg tega za delovanje ne potrebuje konfiguracije in serverja, ampak lahko bere ter piše direktno na navaden disk. Naštete lastnosti ga naredijo zelo primernega za uporabo v mobilnih aplikacijah na praktično vseh platformah.

- **.NET** je programkso ogrodje, ki ga je razvil Microsoft in teče primarno na napravah Windows. Vsebuje ogromno zbirko knjižnic in nam omogoča, da lahko uporabljamo več jezikov hkrati. V praksi to pomeni, da lahko koda, napisanem v enem jeziku, uporablja kodo, napisano v drugih jezikih.

- **GitHub** je odprtokodni sistem za kontrolo različic, ki nam omogoča hranjenje kode na spletu. Z dodatkom je mogoče shranjevati različice neposredno iz Visual studia 2015. Glavna prednost tega početja je, da je koda vedno na varnem ob primeru okvare računalnika ali česar podobnega, prav tako pa lahko ohranimo zadnjo delujočo verzijo aplikacije, če se odločimo za kakšne večje spremembe v razvoju.

V okolju Windows imamo na razpolago različne tehnologije, kot so Silverlight, WPF, UWP idr., s katerimi je mogoče razviti aplikacijo Windows Phone, vendar pa trenutno na tem področju vlada prava zmeda, saj obstaja mnogo nasprotujočih si mnenj na področju razvoja aplikacij Windows phone 8.1, medtem ko se je razvoj na napravah, ki podpirajo Windows 10, usmeril k totalni uporabi UWP-applikacij. Sam sem se odločil za razvoj aplikacije Windows 8.1 universal, vendar mislim, da je za lažje razumevanje pomembno omeniti tudi ostale principe in njihove razlike, saj drugače lahko med razvojem hitro naletimo na nepremostljivo oviro.

2.1 WinRT in Windows RT

Zmeda nastane že čisto na začetku, saj se lahko na spletu kaj hitro zapletemo že s terminoma WinRT in Windows RT. Čeprav gre za zelo različni zadevi, se mnogokrat zasledi, da ju najdemo pod imenom Windows Runetime.

2.1.1 Windows RT

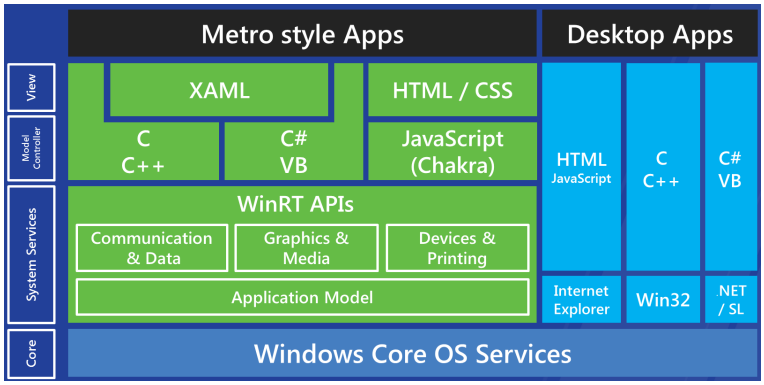
Windows RT je operacijski sistem, izdan sočasno s sistemom Windows 8, s to razliko, da je bil RT zdizajniran za naprave, ki uporabljajo ARM-arhitekturo. To so predvsem mobilne naprave in tablice, ki so jim želeli s tem operacijskim sistemom povečati življenjsko dobo baterije. Windows RT se navzven skorajda ni razlikoval od svojih sodobnikov, je pa bila glavnina sprememb v osrčju programa. To pa je prineslo s seboj nove probleme, saj je zahteval drugačen pristop k razvoju aplikacij, kar je skupaj z nekompatibilnostjo z že obstoječimi napravami Windows pripeljalo do njegovega propada [1].

2.1.2 WinRT

WinRT je, preprosto rečeno, privzet programski vmesnik, ki leži na vrhu operacijskega sistema in je od Windows 8 dalje prevzel vlogo Win32. To seveda ne pomeni, da ga je nadomestil, saj še vedno teče pod vsemi aplikacijam Windows. Služil je kot osnova za izdelavo aplikacij Metro, ki so čez leta doživele mnogo sprememb in jih zdaj poznamo kot UWP-aplikacije. Tečejo v nadzorovanem okolju in potrebujejo uporabnikovo dovoljenje za dostopanje kritičnih točk značilnosti operacijskega sistema ter spodaj ležeče strojne opreme.

2.2 Silverlight

Silverlight je močno razvojno orodje za razvoj interaktivnih bogatih spletnih (Rich Internet Application) in mobilnih aplikacij. Je zastonjski plugin, podprt v ogrodju .NET, in je kompatibilen z mnogo brskalniki, napravami



Slika 2.1: WinRT-arhitektura

ter operacijskimi sistemi[2]. Z njim je torej mogoče ustvariti aplikacije Windows Phone Silverlight, ki tečejo na vseh mobilnikih z operacijskim sistemom Windows 7+, vendar ne na vseh napravah Windows 8+, ki so zgrajene na ARM-arhitkturi. Posledično to pomeni, da ne uporablja WinRT API-jev, kar je prispevalo svoj delež k zatonu Silverlighta. Silverlight 5 je zadnja izdana verzija tega orodja, vendar se je s tem nadaljnji razvoj zaključil.[3] Ohranjajo še podporo in bugfixe, kar pa po mojem mnenju ni dovolj, da bi se splačalo aplikacijo razvijati na tem orodju.

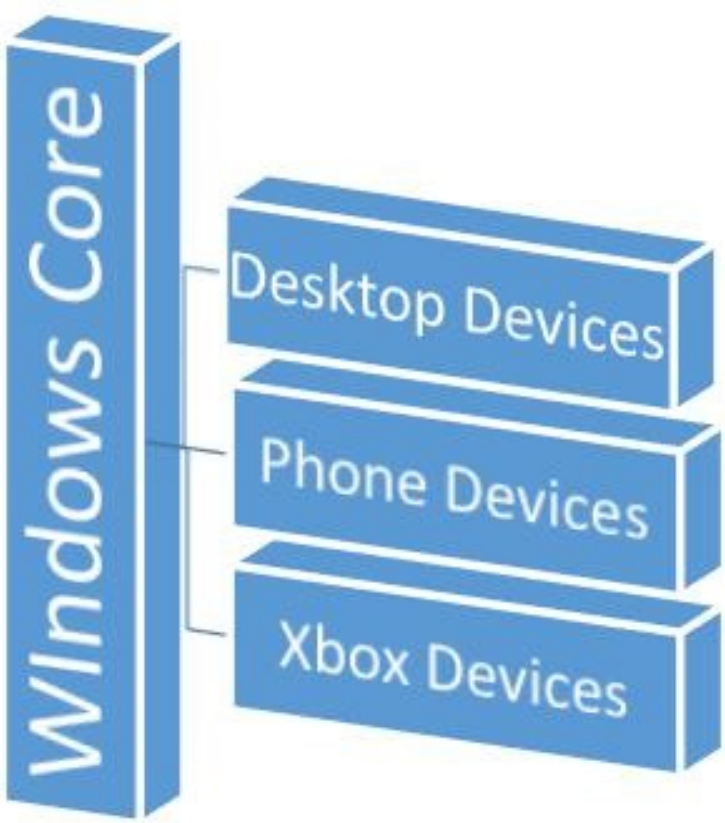
2.3 WPF

Windows Presentation Foundation je računalniški grafični podsistem za delo z uporabniškim vmesnikom v aplikacijah Windows. WPF je motor zadolžen za ustvarjanje, prikaz in upravljanje z uporabniškim vmesnikom, slikami ter multimedijskimi napravami v Windows 7 in kasnejšimi operacijskimi sistemi Windows.[4] Za razliko od Silverlighta je del ogrodja .NET, medtem ko ga Silverlight samo vsebuje. V praksi to pomeni, da so mu na voljo vse funkcionalnost,i ki jih .NET ponuja, vendar pa je omejen samo na okolje Windows, medtem ko lahko Silverlight poganjamo tudi na operacijskih sistemih OSX in Linux. Razlike so lahko zelo subtilne, zato se je treba pred začetkom razvoja

dobro pozanimati, kaj bi od aplikacije sploh radi in komu je namenjena, saj je kasnejše convertanje lahko zelo zamudno ali pa niti ni mogoče. Grobo rečeno je WPF bolj zrela tehnologija, ki nam omogoča večji nabor orodij, medtem ko ima Silverlight večji domet pri napravah, na katerih lahko deluje[5] [6].

2.4 UWP

UWP je trenutno najnovejša aplikacijska arhitektura, ki jo je razvil Microsoft in nam je bila predstavljena s prihodom Windows 10, čeprav njen začetek sega že v Windows 8. UWP je podaljšek sistema WinRT, ki nam je v Windows 8.1 omogočil, da smo lahko naredili aplikacijo hkrati za osebne računalnike in mobilne naprave. UWP je naredil še korak dlje in glavna ideja je, da zgradimo eno aplikacijo, ki deluje na veliki množici naprav, kot so računalniki, tablice, mobilne naprave, Xbox in naprave, ki podpirajo IoT z minimalnimi popravki. UWP cilja na družino naprav namesto operacijskega sistema. Družina naprav identificira API-je, sistemske značilnosti in obnašanja, ki jih lahko pričakujemo od naprave v družini naprav. Osnovni API-ji (WinRT) UWP-ja so enaki za vse naprave Windows, zato bo aplikacija, ki bo uporabljala samo osnovne API-je, delovala na vseh napravah Windows 10. Seveda pa lahko dodamo specializirane API-je za vsako družino naprav z dodatnimi SDK-ji. Vse skupaj nam bistveno zmanjša količino napisane kode. Nova dodatna prednost je, da lahko dodamo napravo na Windows Store, kjer je potem lahko na voljo vsem napravam ali pa samo tistim, ki jih sami določimo. Aplikacije so zapakirane in distribuirane s formatom .AppX, kar nam omogoči zaupanja vreden namestitveni mehanizem in enostavno nalaganje ter posodabljanje. Kljub vsemu pa vse skupaj ni tako rožnato, saj se nad UWP-jem pritožujejo predvsem razvijalci videoiger, za katere je UWP sistem s preveliko tehničnimi omejitvami in označujejo to potezo Microsofta kot zelo agresiven korak k temu, da postane osebni računalnik zaprta platforma[7].



Slika 2.2: UWP-arhitektura

2.5 PRISM

Prism je uradni vodič Microsoftove ekipe za vzorce in najboljše prakse (Microsoft Patterns and Practices Team) za izdelavo "kompozitnih"aplikacij v okoljih/ogrodjih WPF, Silverlight, UWP, Windows 10 in Xamarin forms, vendar je pomembno vedeti, da se prism za vsako izmed naštetih ogrodij razvija neodvisno, in sicer uporabo vzorcev, ki utelešajo pomembnost principov arhitekturnega načurtovanja, kot so seperation of concerns (SoP) in loose coupling. Prism nam pomaga načrtovati in izgraditi aplikacije z uporabo ohlapno povezanih komponent, ki se lahko razvijajo neodvisno, so pa prav tako lahko enostavno ter neopazno integrirane v aplikacijo. Tak tip aplikacije je poznan kot komponentna aplikacija. Kaj je torej vsebuje ta vodič? Prism

za razliko od prej naštetih tehnologij ni samo ogrodje, temveč vsebuje tudi implementacije referenc, obsežno dokumentacijo, mnogo primerov uporabe v obliki manjših projektov in, kar je najpomembnejše, dodatne funkcionalnosti v obliki knjižnic (datotek .dll), ki so del ogrodja. Ogrodje prism nam ponuja implementacijo zbirke načrtovalnih vzorcev, ki so zelo uporabni pri pisanju dobro strukturiranih aplikacij, kot so MVVM, dependecy injection, commands, EventAggregator itd. Poglavitvena funkcionalnost ogrodja je skupna osnovna koda v Portable Class Library (PLC), ki cilja na prej omenjene platforme. Funkcionalnosti, ki pa so za platforme specifične, so implementirane v ustreznih knjižnicah, ki jih uporablja dana platforma. Komu je torej Prism namenjen? Razvijalcem programske opreme na prej omenjenih platformah, ki želijo aplikacijo z več ekrani, bogato uporabniško izkušnjo in prikaz podatkov ter ki poosebljajo pomembno predstavitveno in poslovno logiko. Te aplikacije ponavadi komunicirajo z velikim številom končnih sistemov (back-end systems) in servisi. Pričakuje se, da da se bodo te aplikacije veliko razvijale dalje, skozi daljše časovno obdobje, in sicer kot odgovor na nove potrebe ter poslovne priložnosti. Prav tako je namenjen tudi arhitektom, ki s primeri uporabe vidijo, kako se razvija rešitve za različne platforme in kako naj bi bila videti arhitektura te rešitve z najboljšimi praksami, vodjem razvoja ter analitikom. Celotno ogrodje je narejeno kot paketki, ki jih lahko vzamemo vse ali pa samo majhen delček, ki jih potrebujemo.

Poglavje 3

Razvoj mobilne aplikacije

V tem poglavju bom podrobno predstavil izbiro tehnologij in arhitekture, s katero sem postavil temelje mobilni aplikaciji mFRI. Opisal bom razloge, ki so me vodili h končni odločitvi, in tudi samo delovanje aplikacije ter možnosti za njeno nadgradnjo.

3.1 Arhitektura in tehnologije

3.1.1 Windows 8.1 Universal

Sam sem se oločil za izdelavo aplikacije Windows 8.1 Universal, saj sem začel z izdelavo ogrodja malo po izidu Windows 10 in je bil UWP še popolnoma nova tehnologija, za katero ni bilo na voljo veliko virov. Windows 8.1 Universal tako kot UWP uporablja WinRT-apije, kar pomeni, da aplikacije, razvite za to okolje, delujejo tudi na Windows 10, medtem ko UWP-aplikacije ne delujejo na nižjih verzijah sistema. Začnemo tako, da v Visual Studiu odpremo nov projekt in odpremo vzorec Universal Apps. V projektni rešitvi se nam odprejo trije novi projekti:

1. **Windows 8.1 (Store) project** V tem projektu za zdaj ni nič, ker se mi zdi, da je smisel te aplikacije za zdaj v mobilnosti, česar z aplikacijo za osebni računalnik ne bi dosegli. Vseeno pa nam to ne bi vzelo

veliko časa, saj bi za ta projekt bolj ali manj morali samo napisati oz. popraviti poglede, ki so v projektu Windows Phone 8.1.

2. **Widnows Phone 8.1 project** Vsebuje pretvornike, ki nam prevedejo tipe spremenljivk za potrebe XAML-ja, lastne kontrole, slike in poglede.

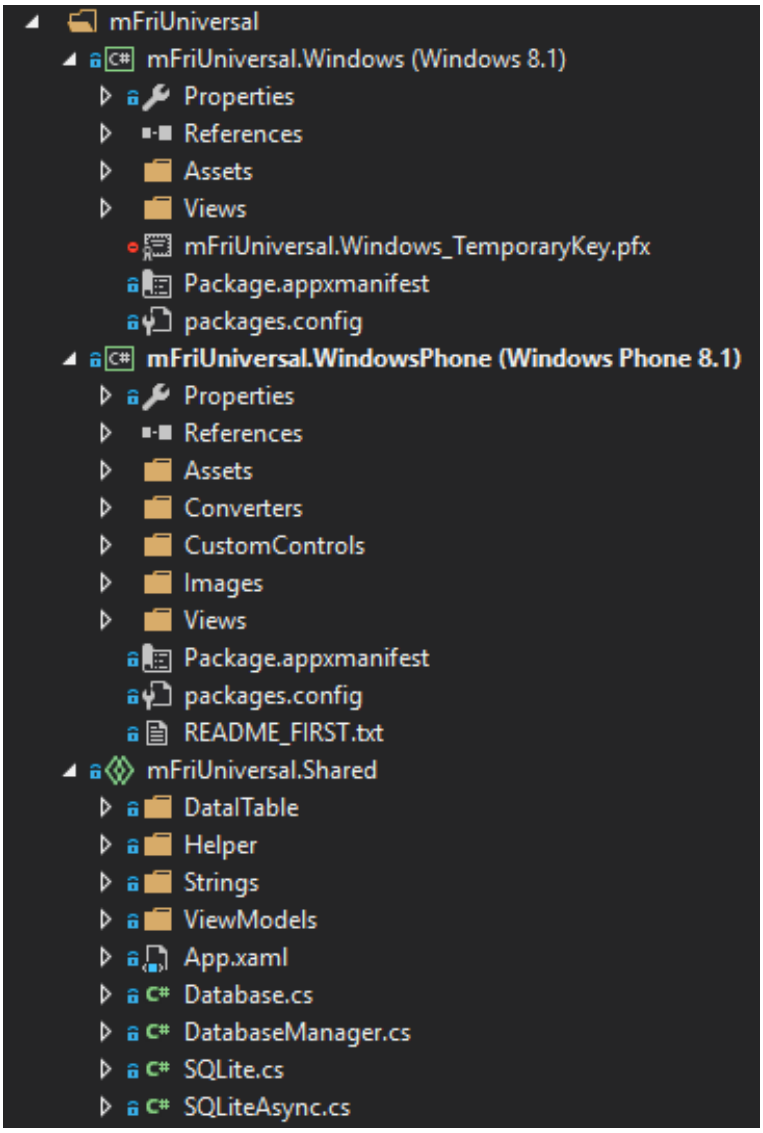
3. **Shared project** Vsebuje implementacijo baze z vsemi pomožnimi razredi, dokumente Resource.resw, ki so v datoteki Strings, ter seveda ViewModele.

Ne moremo poganjati aplikacije Windows Phone na namiznem računalniku in obratno, vendar pa se pod površjem izvaja večinoma ista koda. Namesto da naredimo dve aplikaciji s prekrivajočo se kodo, nam ta model omogoča poganjanje aplikacije na obeh platformah. V projektu Shared je koda, ki jo lahko uporabljata preostala projekta. Projekta Windows Phone in Store seveda uporabljata tudi stvari, ki so specifične za njune platforme. Tipičen primer za to je XAML, ki se ga med drugim uporablja za izdelavo uporabniškega vmesnika in se razlikuje glede na platformo. Zagonska koda aplikacije je v datoteki App.xaml.cs v projektu Shared. V njej so stvari, kot so preverjanje internetne povezave, nalaganje podatkov aplikacije, izbira začetnega pogleda ...

3.1.2 Modularnost

Ogrodje naše mobilne aplikacije sem hotel sestaviti čim bolj modularno, zato naša aplikacija vsebuje poleg treh projektov, omenjenih v prejšnjem poglavju, še šest projektov, ki v obliki knjižnic skrbijo vsak za eno funkcionalnost aplikacije. Moduli služijo kot "Model"v MVVM-arhitekturi in za poslovno logiko našega ogrodja.

Večino omenjenih modulov sem sestavil na podlagi podatkov, ki jih bodo obdelovali. Ti podatki so pridobljeni z javne spletne strani opendata.si in so zapisani v jeziku JSON. Na podlagi teh podatkov sem lahko sestavil ustrezne razrede, ki bodo lahko te podatke ustrezno uporabili. Vsak modul ima poleg že omenjenih tudi glavni razred, ki je zadolžen za klicanje spletnih servisov.



Slika 3.1: Videz projekta Windows 8.1 Universal

- **BusesModule** Skrbi za klice funkcij in spletnih servisov, s katerimi pridobi podatke o avtobusih Ljubljanskega potniškega prometa. Te podatke pretvori v smiselne razrede in z njimi operira po potrebah aplikacije.

- **MenuModule** Podobno kot BusesModule skrbi za klice funkcij in spletnih servisov za pridobivanje podatkov o restavracijah s študentskimi boni. Te podatke tudi shranjuje, saj želimo do njih dostopati tudi, ko internetna povezava ni na voljo.

- **UILogic** Tukaj bi naj bile funkcionalnosti, ki so premajhne za svoj lastni modul, trenutno ima dve glavni funkcionalnosti. Po inicializacije ves čas preverja, ali ima uporabnik na voljo kakršenkoli tip internetne povezave, in to sporoča naprej glavnemu delu aplikacije. Vsebuje pa tudi razred EncryptionHelper, v katerem sta implementirani metodi za kriptiranje in dekriptiranje.

- **MoodleModule** Skrbi za prijavo v FRI-jevo spletno aplikacijo Moodle. Aplikacija najprej pošlje naprej uporabniško ime in geslo, kjer nam ob njunem ujemanju Moodle vrne žeton. Ta žeton lahko potem uporabljamo določeno število časovnih enot, preden poteče. Pošljemo ga zopet na Moodle, ki nam tokrat vrne informacije o strani. S temi informacijami in žetonom pa lahko potem pridobimo podatke o študentu. Za zdaj nam aplikacija samo izpiše ime, priimek, vpisno številko in predmete, ki jih študent obiskuje, medtem ko bi nam za kaj več to morala odobriti fakulteta.

- **BicycleModule** Spet podobna funkcionalnost kot pri BusesModule in MenuModule, le da upravljamo s podatki ljubljanskih mestnih koles (Bicikelj)

- **TimetableModule** Vsebuje funkcije in metode, s katerimi nam omogoča prenos urnika s spleta ter dodajanje in brisanje svojih predmetov. Za razliko od ostalih glavni razred tega modula ni uporabljen zgolj za

pridobivanje podatkov s spleta, ampak tudi aktivno sodeluje pri funkcionalnostih, kot sta brisanje in dodajanje predmetov na urniku.

S tako ločitvijo sem hotel doseči, da bi lahko funkcionalnosti aplikacije bile enostavne za dograditi oz. popraviti, prav tako pa enostavne za implementacijo, saj bi lahko v morebitni novi aplikaciji samo dodali ta modul v obliki datoteke .dll in bi tako dobili delujočo poslovno logiko aplikacije brez posebnega truda. Moduli so med seboj neodvisni, kar pomeni, da manko enega ne vpliva na delovanje celotne aplikacije.

3.1.3 MVVM

Kaj je MVVM

MVVM je zelo popularen arhitekturni vzorec, ki ga je razvil Microsoft za poenostavitev akcijsko usmerjenega programiranja. Je naslednik arhitekturnega vzorca MVP (model view presenter). MVVM je v primerjavi z MVP preformančno slabši, prav tako ga je bistveno težje implementirati, vendar pa se MVVM bolje izkaže pri razširljivosti in vzdrževanju aplikacije, kar je bil tudi eden glavnih ciljev razvoja.

MVVM je v osnovi sestavljen iz treh delov.

Model - model

Model deluje kot neke vrste domenski objekt. V modelu so načeloma podatki ali informacije, s katerimi aplikacija operira. V našem primeru model za restavracijo vsebuje podatke, kot so naslov, meni, odpiralni čas, cena itd. Glavni namen modela je, da informacije samo hrani in nad njimi ne izvaja akcij ali kliče servisov, ki bi podatke spreminjali. Prav tako ni namen, da bi spreminjal besedilo in vplival na njegov videz na ekranu ali pridobival podatke s serverja. Poslovno logiko poskušamo čim bolj ločiti od modela, čeprav je to mnogokrat pravi izziv. Za to želimo dodatne razrede, ki upravljajo z modelom.

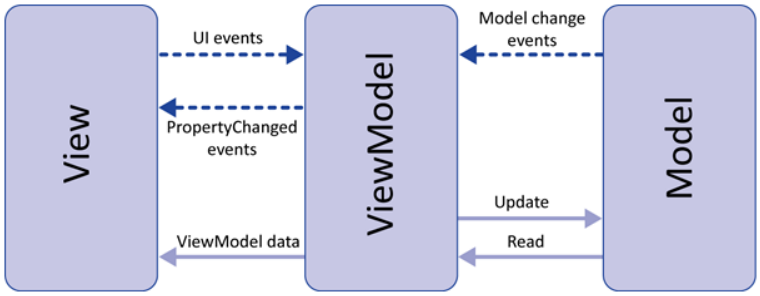
View - pogled

Pogled je najvišji sloj v tej arhitekturi in je namenjen videzu aplikacije ter njenih podatkov na uporabnikovi napravi. Pogled je edina stvar, s katero je končni uporabnik ponavadi seznanjen in s čimer uporabnik interaktira. Pogled omogoča uporabniku akcije, kot so vnašanje podatkov, pritiski gumbov, premiki miške itd., s katerimi vpliva na stanje podatkov v modelu. V okolju .Net ga sestavlja opisni jezik XAML. V MVVM-ju pogled deluje aktivno, kar pomeni, da se za razliko od pasivnega pogleda zaveda dogajanja v modelu in pogledumodelu (kar bom opisal v nadaljevanju). Pogled vsebuje obnašanja, akcije in podatkovne vezi. Čeprav so te ponavadi zmapirane na lastnosti komponente, klice metod in ukaze, je pogled še vedno odgovoren za svoje akcije ter ne preloži vsega dela pogleduModelu. Prav tako pa je treba omeniti, da pogled ni odgovoren za stanje aplikacije, ampak sinhronizira to funkcionalnost s pogledommodelom.

ViewModel - pogledModel

PogledModel je v tem arhitekturnem vzorcu ključnega pomena, ker nam omogoča glavno funkcionalnost, kar je ločitev pogleda od modela. Deluje kot posrednik med pogledom in modelom, tako da jima med seboj ni treba komunicirati. Modela ne zanima, kako so podatki, ki jih poseduje, videti pri pogledu, tako kot pogleda ne zanima, kakšne podatke hrani model, ampak samo ve, kaj mora prikazovati. Tukaj nastopi na vrsto pogledmodel, ki vzame podatke iz modela in jih pripravi tako, da jih bo pogled lahko uporabil, prav tako lahko sprejme podatke iz pogleda, jih obdela in posreduje naprej modelu ali pa se poveže s servisi ter pošlje primerne podatke obema plastema.

Pogled in pogledmodel komunicirata preko sporočil, akcij, podatkovnih vezi ter klicev metod. Prav tako pogledmodel omogoči posreden dostop ne samo do modelov, ampak tudi informacij o stanju in ukazom. Pogled upravlja svoje UV-akcije in jih potem mapira na pogledModel preko ukazov. Modeli in lastnosti (ang. property) na pogledmodelu se vežejo na pogled preko dvosmernih podatkovnih vezi, kar pomeni, da se sprememba na pogledu takoj



Slika 3.2: MVVM-arhitektura

pozna v pogledModelu in obratno.

View in ViewModel

Pogled in pogledmodel komunicirata preko sporočil, akcij, podatkovnih vezi ter klicev metod. Prav tako pogledmodel omogoči posreden dostop ne samo do modelov, ampak tudi informacij o stanju in ukazom. Pogled upravlja svoje UV-akcije in jih potem mapira na pogledmodel preko ukazov. Modeli in pro-pertiji na pogledmodelu se vežejo na pogled preko dvosmernih podatkovnih vezi, kar pomeni, da se sprememba na pogledu takoj pozna v pogledmodelu in obratno.

Zakaj torej MVVM

Glavne prednosti so:

- Da se zgraditi večino aplikacij WPF, Sliverlight in UWP.
- Uporabniški vmesnik lahko dizajniramo neodvisno od razvoja aplikacije, saj je pogled pametno ločen.
- Dovoljuje uporabo testov unit.
- Zelo lahka je ponovna uporaba že uporabljenih komponent tako v sami aplikaciji kot med različnimi projekti.

- Enostavno lahko zamenjamo uporabniški vmesnik brez spreminjanja modelov poslovne logike in ostalih stvari v ozadju.

Glavni kritiki MVVM-ja naj bi bili njegova kompleksnost in strma učna pot, vendar se mi zdi, da ju je zaradi kasnejše preglednosti ter možnih nadgradenj vredno prebroditi.

MVVM v našem ogrodju

Najprej naj omenim, da je MVVM samo vzorec, ki nam kaže, kako naj bi bila aplikacija zgrajena, vendar je končna implementacija še vedno odvisna od razvijalca, in kot tak predstavlja samo del naše rešitve. V praksi to pomeni, da se velikokrat zgodi, da se MVVM-pravila ne uporabljajo čisto dosledno, saj se je treba odločiti med tem, kako močno se želimo držati vzorca in časom, ki ga želimo porabiti, kar je večni predmet debate pri produkcijskem razvoju. Sam sem se poskusil vzorca čim bolj dosledno držati, vendar mi je za izdelavo popolnega MVVM-vzorca zmanjkalo znanja in izkušenj, ki tukaj igrajo veliko vlogo. To se pozna predvsem v tem, da mi nekateri modeli delajo več, kot samo hranijo podatke, jih tudi nekoliko obdelajo, shranijo in kličejo servise. Prav tako pa sem malo zašel iz tega vzorca pri izrisovanju aktivnih točk na zemljevide, saj sem moral določene funkcije pogledModela narediti v pogledu.

3.2 Branje in shranjevanje podatkov

3.2.1 JSON in aplikacijski podatki (App data)

WinRT ponuja uporabniku oz. aplikacijam datoteke, kamor lahko aplikacija shranjuje podatke. Poznamo dva tipa podatkov, uporabniški in aplikacijski, vendar nas zanimajo samo slednji.[**?**]

- **Lokalni oz. Local** Lokalni imenik nam omogoča shranjevanje datotek, povezanih z aplikacijo. Imenik je privatni, kar pomeni, da druge

aplikacije ne morejo dostopati do njega. Imenik se popolnoma izbriše, ko izbrišemo aplikacijo.

- **Plavajoči oz. Roaming** Plavajoči imenik nam omogoča shranjevanje datotek, ki jih sinhronizira z drugimi napravami, ki uporabljajo isti Microsofrov račun ter isti ime paketa v njihovih manifestih. Podatki, shranjeni v tem imeniku, ostanejo v oblaku še nekaj tednov, tudi ko izbrišemo aplikacijo z vseh naprav, kjer smo jo uporabljali.

- **Začasen oz. Temporary** Podatki, shranjeni v tem imeniku, so lahko izbrisani kadarkoli ali na uporabnikovo željo ali ko sistem potrebuje prostor.

- **Lokalni predpomnilnik oz. LocalCache (samo Windows Phone 8.1)** Podobno kot lokalni imenik, z eno pomembno razliko, in to je, da se nikoli ne naredi varnostna kopija datotek na oblaku in bodo vedno iste samo na aplikaciji, na kateri so bile narejene.

Sam sem pri razvoju te aplikacije uporabljal samo lokalni imenik, v katerem sem ustvaril JSON-datoteke in vanje zapisoval podatke, ki sta jih BusesModule in TimetableModule pridobila s spleta. Modula pretvarjata podatke iz datoteke namesto direktno s spleta, saj je pomembno, da po prvi pridobitvi podatkov lahko uporabljamo njune funkcionalnosti tudi brez internetne povezave. Po končani obdelavi podatkov jih ponovno pretvorita v JSON in datoteko prepišeta. Ta način sem ubral, ker se mi je zdel najbolj enostaven, saj bi ob uporabi SQLitea moral podatke iz JSONa pretvoriti v razrede in tabele, tako pa sem lahko en korak preskočil, medtem ko funkcionalnosti nisem izgubil.

3.2.2 SQLite

Pri shranjevanju sem uporabljal tudi SQLite, saj so nastavitve idealne za baze, kajti vsebujejo tabelo samo z eno vrstico, ki hrani vse bistvene vrednosti, od katerih je odvisno uporabniku prijaznejše delovanje aplikacije. Prav

tako sem SQLite uporabil pri shranjevanju avtobusnih postaj, saj zaradi pomembnosti ažurnih podatkov ni imelo smisla hraniti podatkov v datoteko. Ker hočemo vedno najnovejše podatke, je bilo dovolj, da sem shranil ime in številko postaje ter številko avtobusa.[8]

SettingsDataItem	
int?	SettingsId
int	MenuRadius
bool	RememberUsernamePassword
bool	RememberStudentNumber
string	StudentNumber
string	Username
string	Password

- **SettingsId:** Id tabele
- **MenuRadius:** Število, ki nam pove, v radiusu kolikih kilometrov od naše lokacije naj nam prikazuje restavracije s študentskimi boni
- **RememberUsernamePassword:** Vrednost čheckboxa", ki nam pove, ali si naj zapomnimo uporabniško ime in geslo
- **RememberStudentNumber:** Vrednost čheckboxa", ki nam pove, ali si naj zapomnimo vpisno številko študenta
- **StudentNumber:** Vpisna številka študenta
- **Username:** Uporabniško ime
- **Password:** Geslo

StationDataItem	
int?	StationDataItemId
string	Name
string	StationNumber
string	BusNumber

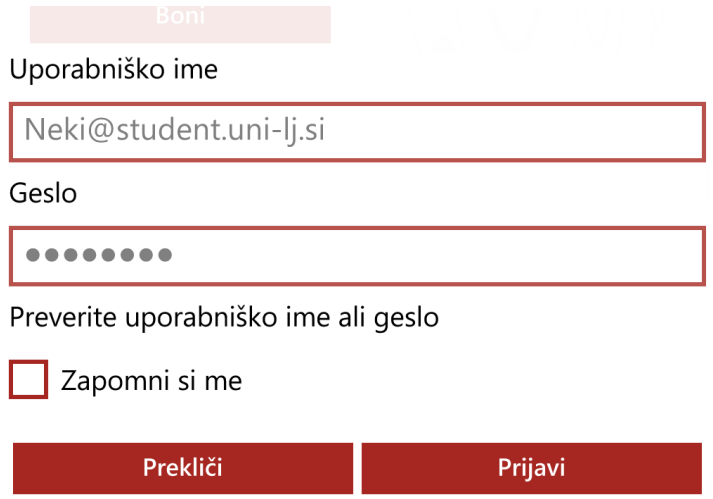
```
<Button Grid.Row="1" Grid.ColumnSpan="2" Style="{StaticResource ButtonFriThemeWh}"
        x:Uid="ButtonSearch" Command="{Binding Path=GetBuses}" HorizontalAlignment="Stretch"/>
```

Slika 3.3: Primer kontrole v XAML-ju

- **StationDataItemId:** Id tabele
- **Name:** Ime postaje
- **StationNumber:** Številka postaje
- **BusNumber:** Številka avtobusa

3.3 Uporabniški vmesnik

Uporabniški vmesnik je narejen v programskem jeziku XAML, kar je značilno tudi za aplikacije Silverlight, WPF in UWP. XAML je opisni jezik, narejen po standardih XML-jezika, in v naši aplikaciji predstavlja View. Kontrola se v XAMLu vedno začne z ≪", ki mu sledi tip objekta, čigar instanco želimo ustvariti, nato pa sledijo parametri, ki jim lahko direktno nastavimo vrednosti ali pa jih povežemo z v našem primeru z ViewModelom. Kontrolo zapremo z ≫"ali pa "/¿", odvisno ali smo uporabili samostoječo kontrolo, kot je gumb, ali pa kontrolo z vsebino, kot je Platno, kot je primer na sliki 3.3. Slika prikazuje tudi primer klicanja funkcije s parametrom Command in Style. Prvi nam omogoča klicanje funkcije, ki je v ViewModelu, Style pa nam omogoča, da v datoteki App.xaml oblikujemo ta element in potem ta stil uporabljamo v vseh ostalih datotekah .xaml. Za razvijalca to pomeni, da lahko potem enostavno zamenja celotno podobo aplikacije, saj lahko vse spremeni v eni datoteki, namesto da išče vsak element posebej in ga popravlja. Tak tip izdelave uporabniškega vmesnika je precej enostaven in hiter, vendar se lahko pri kompleksnejših zadevah ta krivulja zelo hitro dvigne.



Na sliki je pojavno okno ob povezavi aplikacije z učilnico.

Slika 3.4: Primer lastne kontrole

3.3.1 Lastne kontrole

Z malo naprednega znanja lahko v XAML-ju naredimo tudi lastne kontrole, ki jih lahko potem uporabljamo enako kot navadne kontrole pri izdelavi pogleda. Izdelava lastne kontrole se v osnovi ne razlikuje preveč od ustvarjanja pogleda, saj namreč naredimo ravno to. Ustvarimo nov, manjši pogled, ki je spet skupek nekih osnovnih kontrol. Od navadnega pogleda se razlikuje po tem, da moramo kontrolam, ki jih mislimo uporabljati, spremeniti odvisne lastnosti (ang. dependency property). To se napiše ponavadi v kodi v ozadju in je potrebno, da lahko potem naši kontroli vežemo ukaze in vrednosti iz ViewModela. Lastne kontrole so v naši aplikaciji pojavna okna, ki so nam v resnici ves čas na voljo, ampak se pokažejo šele, ko neka akcija sproži, da se jim spremeni vidljivost.

3.4 Večjezičnost

V aplikacijsko ogrodje sem vgradil tudi večjezičnost, ki trenutno omogoča izbiro med angleškim in slovenskim jezikom. V shared imeniku sta dve datoteki Resources.resw, ki vsebujeta vrednost spremenljivke in spremenljivko s parametrom, na katerega se ta veže. Primer te uporabe je na sliki 3.5, kjer lahko vidimo parameter x:Uid, ki mu določimo za vrednost spremenljivko iz Resources.resw datoteke, ta pa nam potem na zaslon izpiše njeno vrednost. Tak način nam omogoča hitro in enostavno dodajanje novih jezikov, saj moramo samo dodati novo datoteko .resw z enakimi spremenljivkami ter prevedemo prejšnje vrednosti. Med razvojem nas ta način nastavljanja precej upočasnjuje, vendar pa je to stalna praksa v delovnem okolju, saj je kasnejša implementacija večjezičnosti praviloma bolj zamudna. Aplikacija na začetku sama izbere, katero datoteko Resourece.resw bo uporabljala glede na nastavitve mobilne naprave, vendar lahko uporabnik sam naknadno spremeni jezik v nastavitvah.

Name	Value
AddCourseButton.Content	Dodaj
AllBuses	Vsi
BicikeljPageTitle.Text	Bicikelj
BusesPageTitle.Text	Trola
ButtonRefresh.Content	Osveži
ButtonSearch.Content	Išči

Slika 3.5: Videz datoteke Resource.resw

3.5 Testiranje

Zelo pomemben del razvoja aplikacije je tudi testiranje. Microsoft nam ponuja na voljo emulatorje, s katerimi lahko simuliramo okolje operacijskega sistema Windows phone. Emulator nam omogoči poganjanje aplikacije v

primernem okolju in hkrati tudi možnost za razhroščevanje. Na žalost pa so z eno posodobitvijo emulatorji prenehali delovati, nakar sem moral začeti razvijati ogrodje na dejanski mobilni napravi. To je prineslo ogromne prednosti, saj se je izkazalo, da so določene stvari, ki pred tem niso delovale, bile zgolj krivda emulatorja, slabost pa je ta, da kar naenkrat nisem mogel več preverjati videza na različnih velikostih ekrana.

3.6 Varnost

Z varnostjo se v aplikaciji soočamo predvsem pri prijavi v sistem Moodle in kriptiranju shranjenega gesla v bazi. Za varnost pri prijavi v sistem Moodle poskrbi sistem že sam, medtem ko sem za varnost baze moral poskrbeti sam. Aplikacija omogoča, da si zapomni uporabniško ime in geslo uporabnika, kar pomeni, da se v bazo shrani geslo iz učilnice (kasneje morda tudi študentski informacijski sistem), zato je pomembno, da jo vsaj malo zaščitimo. Prva stvar, ki sem se je naučil ob iskanju primerne kriptografske metode, je, naj je ne poskušamo napisati sami. Funkcijo sem se torej odločil poiskati na internetu, kjer pa sem naletel na mnogo različnih primerov implementacije. Našel sem relativno preprosto enkripcijsko metodo, ki uporablja napreden enkripcijski standard (AES) oz. njegovo implementacijo iz okolja .NET. [10]

AES je zelo popularen simetričen enkripcijski algoritem, kar pomeni, da za kriptiranje in dekriptiranje uporablja enak ključ. Grobo rečeno, AES-algoritem izračuna število ponovitev in nato v vsaki ponovitvi zamenjuje bloke bytov, premika vrstice in meša stolpce. Moja implementacija algoritma ni najboljša, saj naj bi po standardih uporabljal drugačen ključ za vsako napravo, poleg tega pa je tudi zapisan v kodi. Kljub temu pa mislim, da to ne bi smelo predstavljati problemov, saj mislim, da je za zdaj uporabnikovo geslo vseeno zaščiteno dovolj, da bi napadalca odvrnilo od poskusa razbitja zaščite.

Naj še omenim, da moje baze ni bilo treba posebej zaščititi pred SQL-vdori, saj sem uporabljal LINQ. LINQ v povezavi s SQL-jem uporablja .NE-

Tov razred SQLParamaters, ki uporabnikove vnose spremeni v parametre namesto v sestavljen niz.

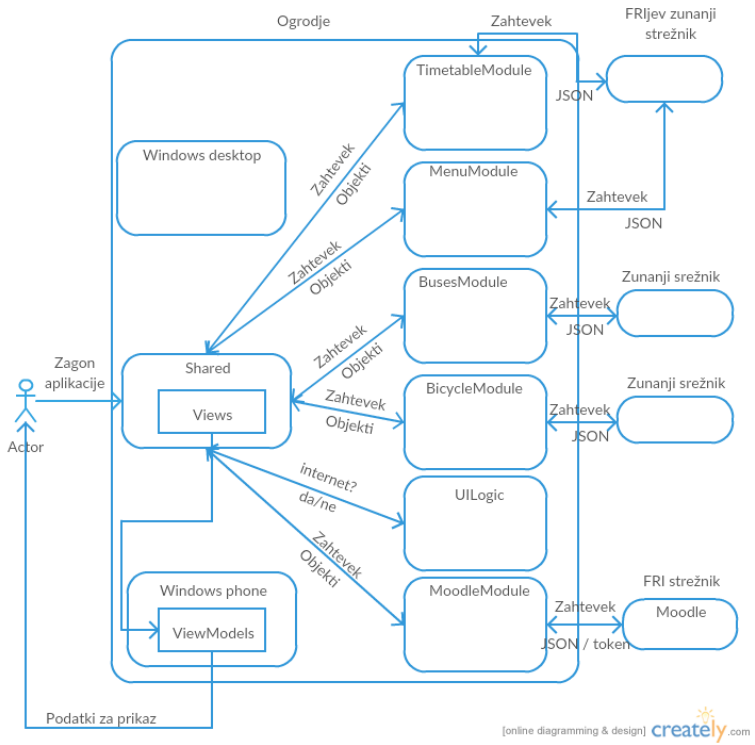
Poglavje 4

Uporaba in delovanje aplikacije

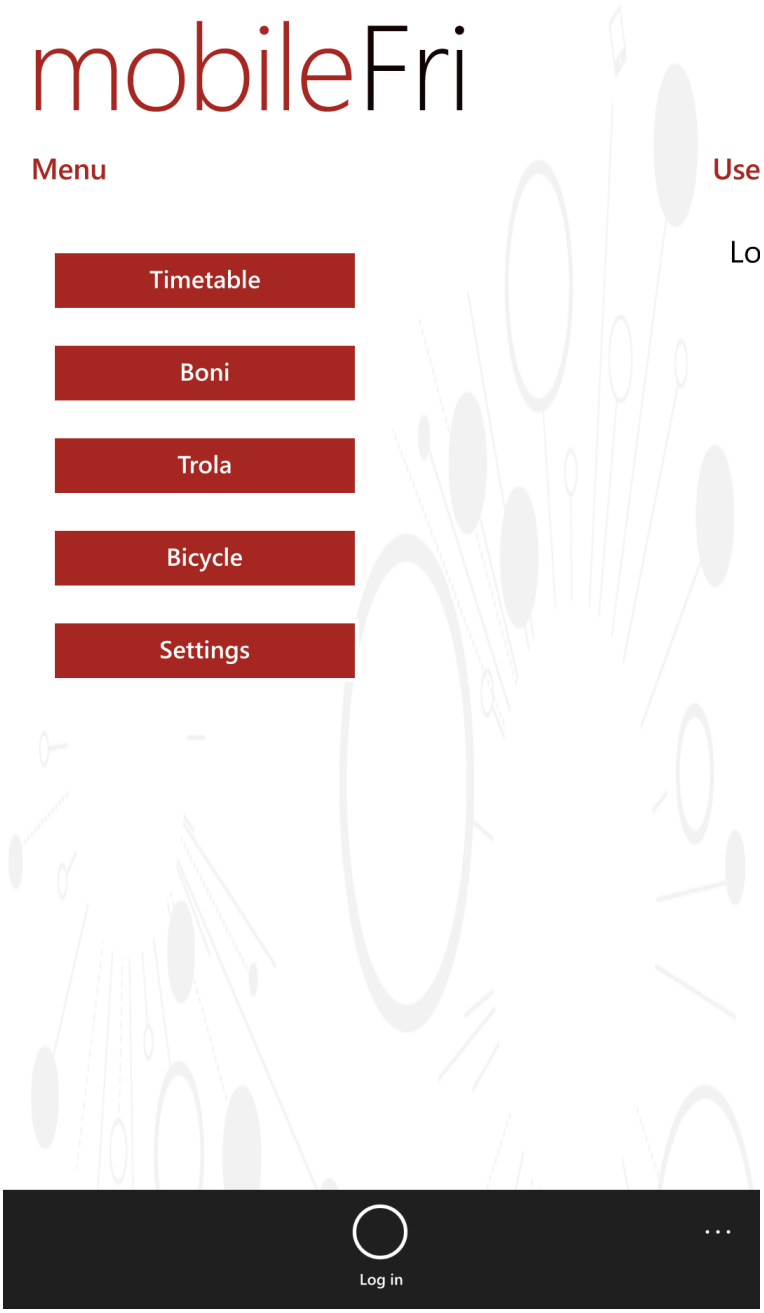
Ob zagonu aplikacije se najprej sproži serija ukazov in funkcij, ki pripravijo našo aplikacijo na delovanje. Za to je zadolžena datoteka App.xaml.cs, ki je v našem projektu Shared. Aplikacija najprej ustvari povezavo z bazo in ustvari tabele. Ob prvem zagonu se te tabele tudi napolnijo z vnaprej pripravljenimi vrednostmi, ki jih lahko potem uporabnik po mili volji spreminja. Sledita preverjanje internetne povezave in aktivacija funkcije, ki bo to celoten čas trajanje aplikacije počela za nas vsako sekundo. Če smo z internetom že povezani, se izvede tudi klic servisa, ki nam ustvari datoteko za hranjenje podatkov in jo tudi napolni s podatki o študentski prehrani. Tako lahko dostopamo do teh podatkov tudi, ko nimamo internetne povezave.

4.1 Glavna stran

Po izvedenem zagonu pristanemo na prvi strani, kjer so trenutno povezave na ostale poglede (v obliki gumbov), gumb za prijavo v Moodle in razni podatki o fakulteti. Pritisk na gumb nazaj bo na tej strani povzročil izhod iz aplikacije, saj je ta stran najvišje na drevesni strukturi. Na drugih straneh bi nas zgolj vrnilo nazaj na glavni pogled. Pri tem je treba omeniti tudi, da gumb nazaj vrže en pogled nazaj, saj je del tiste strani, na kateri je bil odprt, in ne nov pogled.



Slika 4.1: Diagram delovanja aplikacije



Na sliki je začetni meni, od koder lahko navigiramo po aplikaciji. Omogoča pa nam tudi povezavo z učilnico in pregled informacij.

Slika 4.2: Glavna stran aplikacije

4.2 Urnik

V tem pogledu je urnik, ki si ga mora uporabnik ob prvi uporabi naložiti s spleta. Ob pritisku na gumb se pokaže pojavno okno, kamor vnesemo vpisno številko. Aplikacija nato pokliče servis, ki s spletnega urnika prebere in priredi podatke, s katerimi se potem napolni urnik. V primeru prekrivanja se pri vsakem elementu, na katerega to vpliva, izpiše dodatna vrstica, ki nas na to opozori. Urnik prav tako omogoča brisanje že obstoječih predavanj/vaj ali pa dodajanje novih. Za prenos urnika je potrebna povezava z internetom, kasneje pa za uporabo urnika to ni več potrebno, saj se vsi podatki shranijo v tekstovno datoteko, iz katere se ob kasnejši uporabi tudi preberejo.

4.3 Boni

S tem pogledom omogočimo uporabnikom (študentom) najenostavnejši dostop do podatkov o študentski prehrani. Ob prihodu na ta pogled se nam najprej prikažejo podatki o študentski restavraciji FRIK, na voljo pa sta tudi zemljevid z lokacijami restavracij in njihov iskalnik. Na zemljevidu so v obliki točk prikazane vse restavracije, ki so v naši bližini. Ob pritisku na specifično točko se nam v pojavnem oknu izpišejo podrobnosti o restavraciji, kot so cena, meni, odpiralni časi ... Te se od restavracije do restavracije razlikujejo. Iskalnik pa nam omogoča, da lahko pogledamo enake informacije za restavracijo, ki smo jo vnesli v iskalnik. Ker so rezultati prikazani v seznamu, bo iskalnik prikazal vse restavracije, ki vsebujejo iskani niz. Za prvo uporabo je potrebna povezava z internetom, sicer podatki niso dosegljivi. Ob nadaljnji uporabi aplikacije brez internetne povezave se podatki vseeno prikažejo, vendar niso ažurni, kar pa lahko uredimo z gumbom "osveži".

4.4 Trola

V tem pogledu lahko uporabnik ob vzpostavljeni internetni povezavi poišče prihode in odhode avtobusov za izbrano postajo. Pogled ima na voljo dve

funkcionalnosti, med katerima se prestavljamo s pritiskom oz. z drsenjem po pogledu levo ali desno. Ob odprtju tega pogleda nas pričaka osnovna funkcionalnost, kot je iskanje avtobusov. V meniju izberemo številko avtobusa in vnesemo številko ali ime postaje ter pritisnemo na gumb 'išči'. Rezultat tega dejanja je seznam prihodov vseh želenih avtobusov na izbrano postajo. Ob pritisku na katerega izmed elementov seznama se nam prikaže pojavno okno, s katerim lahko postajo shranimo. Tako pa pridemo do druge funkcionalnosti tega pogleda. Drugi seznam vsebuje vse shranjene postaje in se osveži vsakič, ko pridemo na pogled. Tako si lahko postaje in avtobuse, ki jih pogosto obiskujemo, shranimo, ali pa po potrebi tudi brišemo. Ob prvem zagonu se dodata postajališči Živalski vrt in Viško polje, ki sta najbližje fakulteti.

4.5 Bicikelj

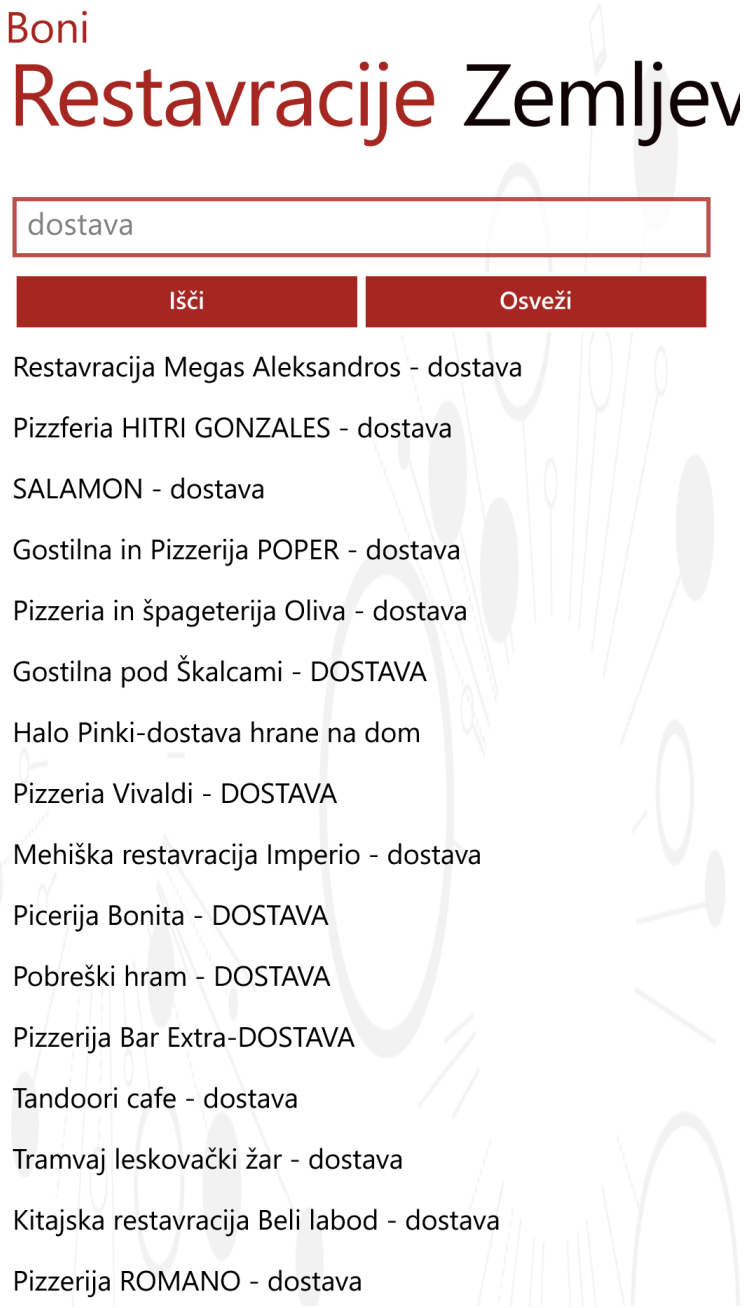
Pogled vsebuje zgolj zemljevid, na katerem so v obliki pinov izpisane kolesarske postaje. S pritiskom nanje se nam odpre pojavno okno, kjer so informacije o količini prostih in vseh koles. Uporabnik se lahko na podlagi tega torej odloči, katere postaje so mu bolj v interesu, saj lahko hitro vidi, ali je postaja polna ali pa koles ni na voljo. Zaradi velike potrebe po ažurnih podatkih tega nikamor ne shranjujemo. Pogled torej dela samo ob prisotnosti internetne in GPS-povezave.

4.6 Nastavitve

V tem pogledu so nastavitve aplikacije, ki pa jih za zdaj ni veliko. Tukaj si lahko uporabnik izbere jezik aplikacije in radius prikazovanja restavracij v pogledu Bon, ter kolesarskih postaj v pogledu Bicikelj. Radius je predstavljen v kilometrih in ima za središče naše GPS-koordinate. Vsebuje tudi gumb za shranjevanje nastavitev in gumb za njihovo poenostavitev.

4.7 Možnosti nadgradnje

Celoten pristop k razvoju tega ogrodja je bil namenjen temu, da služi kot temeljni kamen k nadaljnjemu razvoju. Že med samim razvojem sem slišal mnogo idej za nadaljnji razvoj, kot so dodajanje ponudb študentskega dela za računalničarje, dodajanje reklam, prijava na izpite, obvestila o rokih za oddajo domačih nalog oz. nove domače naloge ... Vse omenjene predloge, mislim, bi bilo ob primernih servisih trivialno dodati v samo aplikacijo. Prav tako pa sem pretvoril svojo aplikacijo Windows 8.1 Universal v UWP z vsemi funkcionalnostmi (vendar iznakaženim grafičnim uporabniškim vmesnokom), ki omogoča poleg dodatnih možnosti tudi dodatne ideje za razširitve, na katere zdaj niti ne pomislimo, saj, kot sem prej omenil v drugem poglavju, jo lahko ob primerni modifikaciji uporabljamo praktično kjerkoli.

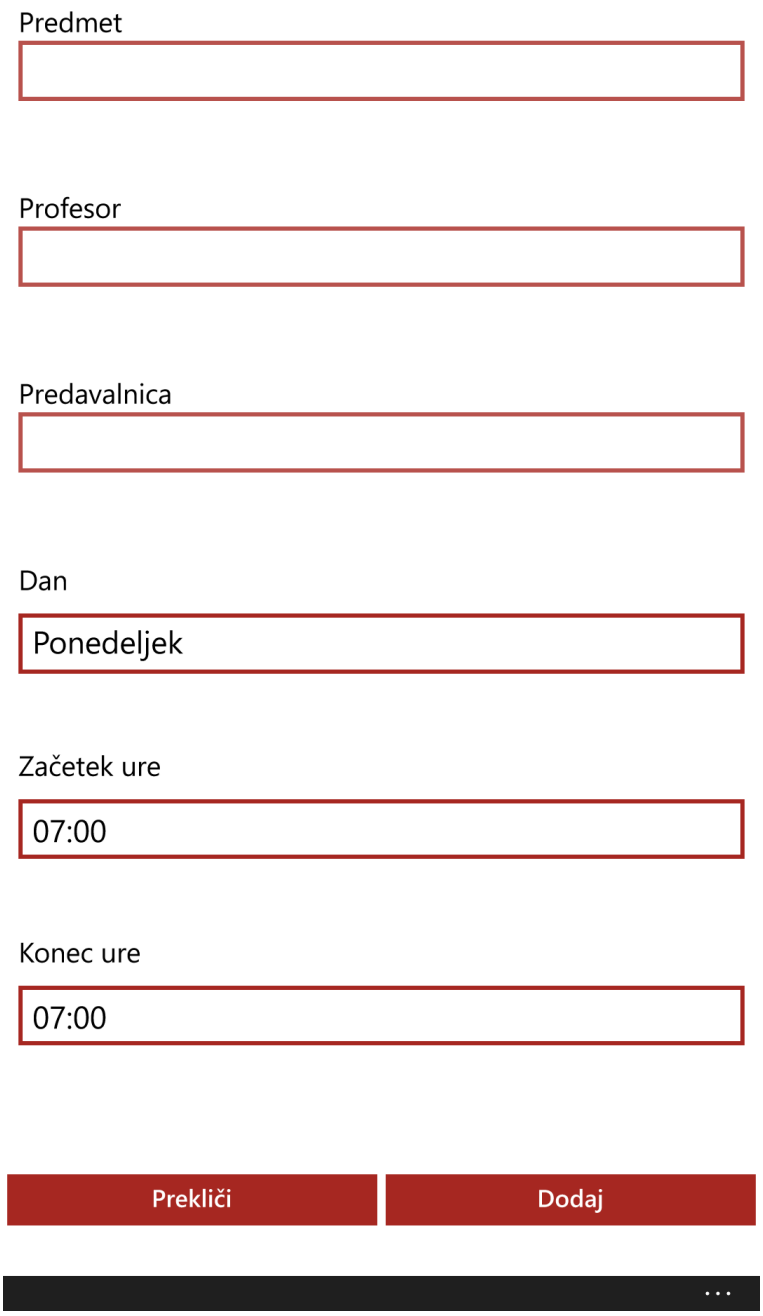


Slika prikazuje seznam vseh restavracij, ki ponujajo študentske bone. S klikom nanjo lahko izvemo odpiralni čas, meni, ceno ...

Slika 4.3: Pogled menija

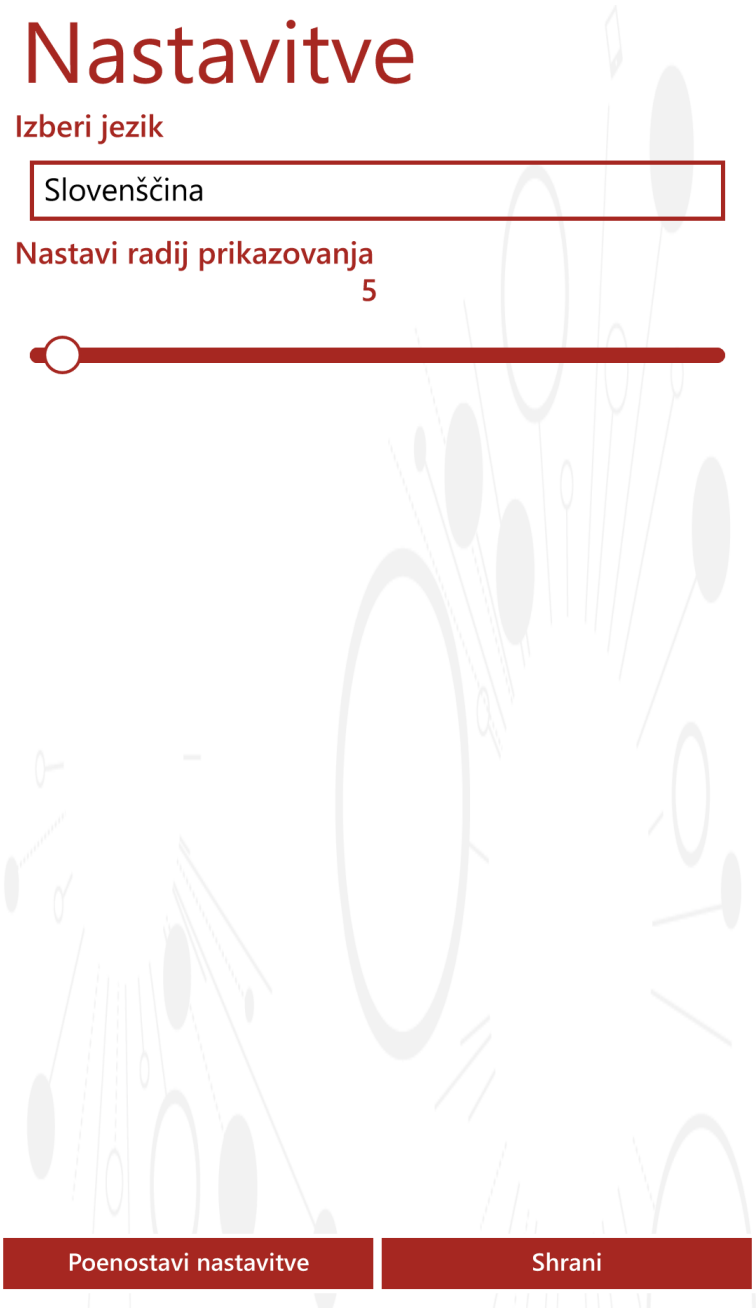
Slika 4.4: Pogled urnika

Slika prikazuje urnik, po katerem lahko navigiramo po dnevih, dodajmo in brišemo predmete ali jih potegnemo s spleta s pomočjo vpisne številke.



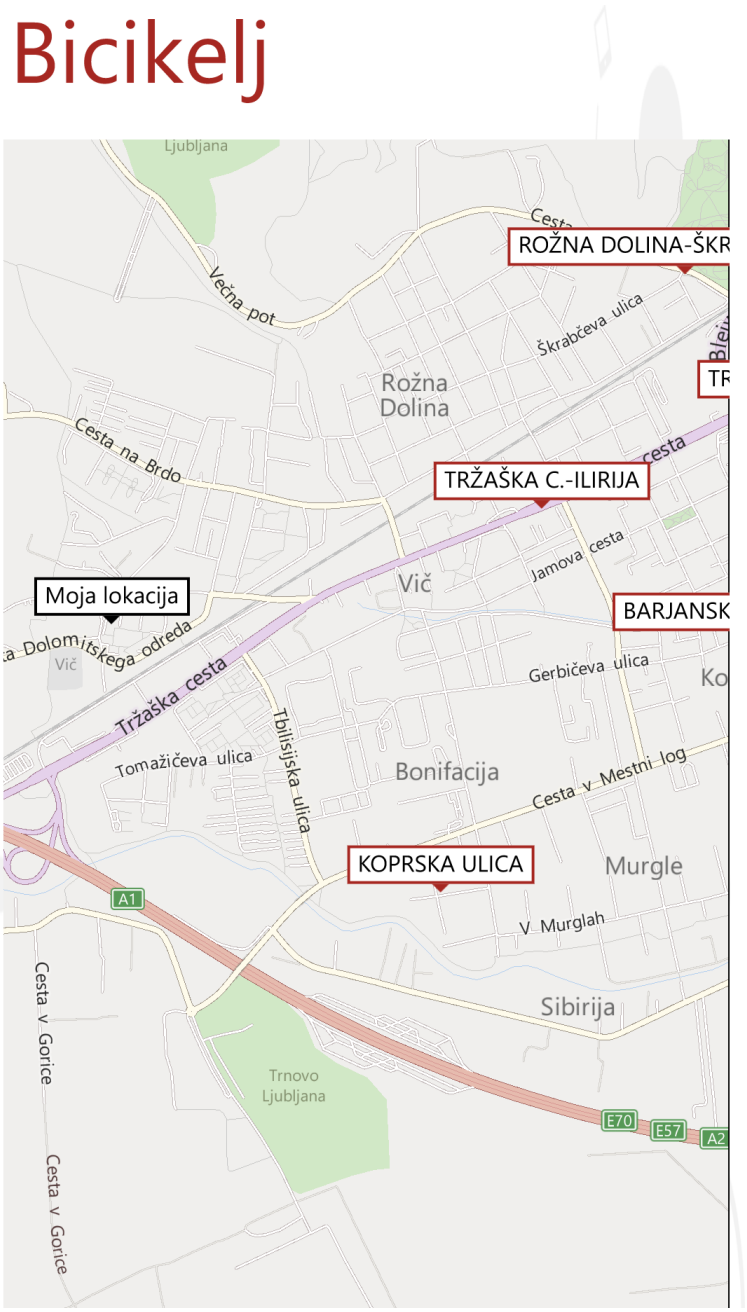
Slika prikazuje okno za dodajanje novega predmeta na urnik.

Slika 4.5: Dodajanje predmeta



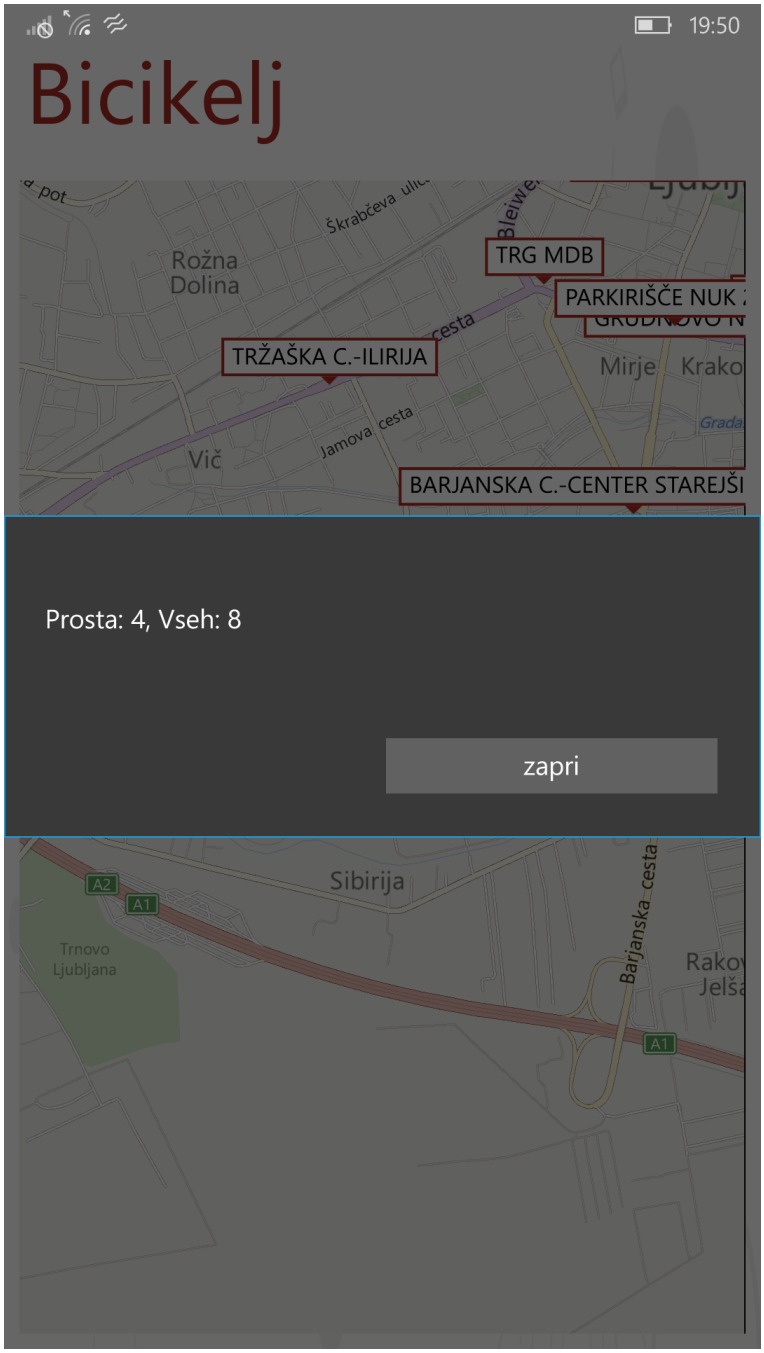
Slika prikazuje nastavitve aplikacije, kjer sta za zdaj na voljo zgolj jezik in radiuis obsega okoli naše GPS-lokacije.

Slika 4.6: Pogled nastavitev



Slika 4.7: Pogled Bicikelj

Slika nam prikazuje našo lokacijo in lokacijo postajališč Bicikelj. Podobno funkcionalnost ima tudi pogled Meni.



Na sliki vidimo prosta kolesa in količino vseh koles na izbranem postajališču Bicikelj.

Slika 4.8: Prikaz podrobnosti

Poglavje 5

Sklep

Moram priznati, da se ob izbiri teme nisem točno zavedal, v kaj se spuščam, saj sem se odločil za razvoj aplikacije Windows phone ravno v času, ko je na trg prišel Windows 10 in z njim tudi UWP. Zanj se je Microsoft odločil, da bo postal njihov absolutni način razvoja aplikacij, zato tudi usmerjajo vse v to.

Zaradi pomanjkanja podatkov, kaj UWP sploh je, dokumentacije in ne nazadnje tudi primerov razvoja sem se odločil, da bom postavil ogrodje za aplikacije Windows 8.1 Universal. Kasneje se to ni izkazalo za preveč slabo odločitev, saj je princip programiranja ostal bolj ali manj enak, zato tudi nisem imel prevelikih težav s pretvarjanjem svoje aplikacije v UWP.

Mislim da je tudi pomembno da poznamo razliko med različnimi Microsoftovimi tehnologijami, saj obstaja na spletu mnogo nasprotujočih si mnenj in tudi napačnih. Predvsem nam je to prišlo prav pri odločitvi za kaj bomo sploh razvijali, čerpav mislim da to zaradi UWPja ne bo več predmet debate, kot tudi pri razhroščevanju saj lahko dosti hitreje prefiltriramo koristne vire od nekoristnih.

Izkazalo se je tudi, da je bila izbira arhitekturnega vzorca MVVM pravilna odločitev, saj je zelo pogosta uporaba v aplikacijah Windows phone. Kljub precej strmi učni krivulji mislim, da je vredno potrpeti, saj je razvoj, ko zadevo enkrat razumemo, zelo enostaven in naraven. Kljub temu pa ne

gre čisto brez težav, saj bi, če bi se hoteli dosledno držati MVVM-vzorca, potrebovali bistveno več izkušenj. Tudi če najdemo primer na spletu, obstaja velika možnost, da bo napisa v kateri izmed preostalih tehnologij, opisanih v tem diplomskem delu. Problem najlažje rešimo tako, da preprosto včasih malo zapostavimo MVVM in kodo prestavimo v View.

Če dodamo tej aplikaciji še modularnost in možnost spreminjanja skoraj celotnega videza ogrodja, mislim, da mi je uspel zastavljeni cilj, ki je obsegal izdelavo funkcionalne aplikacije z ogrodjem, ki bi bilo naslednikom razumljivo in enostavno za implementacijo novih funkcionalnosti.

Literatura

[1] Microsoft MSDN [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/windows/apps/jj681690(v=vs.105).aspx [Dostopano 18. 8. 2016].

[2] Microsoft [Online]. Dosegljivo: https://www.microsoft.com/silverlight/. [Dostopano 18. 8. 2016].

[3] Microsoft Support Lifecycle [Online]. Dosegljivo: https://support.microsoft.com/en-us/lifecycle?C2=12905. [Dostopano 18. 8. 2016].

[4] C-sharpcorner [Online]. Dosegljivo: http://www.c-sharpcorner.com/blogs/what-wpf-is1. [Dostopano 18. 8. 2016].

[5] Microsoft MSDN [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/ff921107(v=pandp.20).aspx. [Dostopano 18. 8. 2016].

[6] Microsoft MSDN Blogs [Online]. Dosegljivo: https://blogs.msdn.microsoft.com/jennifer/2008/05/06/when-should-i-use-wpf-vs-silverlight/. [Dostopano 18. 8. 2016].

[7] Microsoft MSDN Blogs [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp. [Dostopano 20. 8. 2016].

[8] SQLite [Online]. Dosegljivo: https://www.sqlite.org/about.html [Dostopano 20. 8. 2016].

[9] Microsoft MSDN [Online]. Dosegljivo: https://msdn.microsoft.com/library/windows/apps/jj553522.aspx [Dostopano 20. 8. 2016].

[10] StackOverflow [Online]. Dosegljivo: http://stackoverflow.com/questions/10168240/encrypting-decrypting-a-string-in-c-sharp [Dostopano 5. 9. 2016].

[11] CodePlex [Online]. Dosegljivo: https://compositewpf.codeplex.com/ [Dostopano 20. 8. 2016].

[12] Github Prism [Online]. Dosegljivo: https://github.com/PrismLibrary/Prism [Dostopano 21. 8. 2016].